University *of* Exeter

Optimizing Communication Pattern in the Mandelbrot Program

Author
Sina Ataei

High Performance Computing

College of Engineering, Mathematics and Physical Sciences

March 2023

Introduction

In this report, we describe the modifications made to the communication pattern in the manager-worker version of the Mandelbrot program to optimize its parallel performance. Specifically, the original code, which used MPI_Reduce to communicate results at the end of the calculation, was modified to have worker processes send their results to the manager process after each column was calculated. We then present the results of our scaling study carried out on the Isca HPC system.

Modifications to the program

The communication pattern in the original version of the manager-worker Mandelbrot program was modified so that worker processes send their results to the manager process after each column has been calculated instead of calling *MPI_Reduce* at the end of the calculation. This was achieved by creating buffer space for sending and receiving data of the size of one column in the complex plane plus two extra values. The message buffer is initialized to contain zeros for all elements except the column index and the worker rank.

```
   // Message buffer for Manager/Worker communications
  int res_buff[N_IM + 3];   // calculated column
  const int col_idx = N_IM + 1; // index of the column calculated by worker
  const int proc_rank = N_IM + 2; // Rank of the worker that calculated the
      column

  // Initialise the message buffer
  res_buff[col_idx] = -1;
  res_buff[proc_rank] = myRank;
  for (i = 0; i < N_IM + 1; i++){
     res_buff[i] = 0;
  }
```

The case of the initial handout where there is no data to be sent back to the manager process was handled by setting the i value of the column to the missing value of -1.

The point-to-point communication calls were then modified so that worker processes send their results back to the manager process after calculating each column. The i value of the computed column, rank of the computing process, and the computed data were packed into a single buffer, which was sent to the manager process.

```
// Worker Processes
    // Send request for work along with the calculated column
    MPI_Send(&res_buff, N_IM + 3, MPI_INT, 0, 100 + myRank, MPI_COMM_WORLD);
    // Receive i value to work on
    MPI_Recv(&i, 1, MPI_INT, 0, 100, MPI_COMM_WORLD, &status);

    res_buff[col_idx] = i;
    calc_vals(i);

    // pack the calculated column into the message buffer
    for (j = 0; j < N_IM + 1; j++){
        res_buff[j] = nIter[i][j];
    }
```

In the manager process, the for loop that hands out work to worker processes was modified to receive the message buffer containing the calculated column and worker rank. If the column index is valid, the received column is stored in the nIter array. After storing the received column, the manager process sends the next column index to the requesting process. Finally, the call to *do_communication* was commented out.

```
// Manager process
   // Receive request for work
          MPI_Recv(&res_buff, N_IM + 3, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
              MPI_COMM_WORLD, &status);

          nextProc = res_buff[proc_rank];
          int calc_col = res_buff[col_idx];

          // Check if the column index is valid
          if (calc_col >= 0)
          {
              // store the received column in the nIter array
              for (j = 0; j < N_IM + 1; j++)
              {
                  nIter[calc_col][j] = res_buff[j];
              }
          }

          // Send i value to requesting process
          MPI_Send(&i, 1, MPI_INT, nextProc, 100, MPI_COMM_WORLD);
```

Scaling study

The modified code improves the communication pattern by allowing worker processes to send their results to the manager process after each column calculation instead of waiting until the end of the calculation to use MPI_Reduce. This reduces the amount of data that needs to be communicated at once and allows the manager process to store the results in the nIter array as soon as they are available.

We carried out a scaling study using the Isca HPC system with N_RE=12000 and N_IM=8000. The I/O was switched off (by setting doIO=false) for this study. We ran the original program using 2, 4, 8, 12, and 16 MPI processes on one node, and 32 MPI processes on 2 nodes. We also ran the modified program using the same number of MPI processes.

$$S_N = \frac{T_0}{T_N} \tag{1}$$

We then calculated the parallel speed-up data using equation 1 for the original and modified versions of the program, assuming that the serial run time $T_0 = 90.985$ seconds. The results of our scaling study are shown in Fig 1.

Assessment of results

As shown in Fig 1, the modified version of the program performs significantly better than the original version with 12 processes or more. The speed-up for the modified

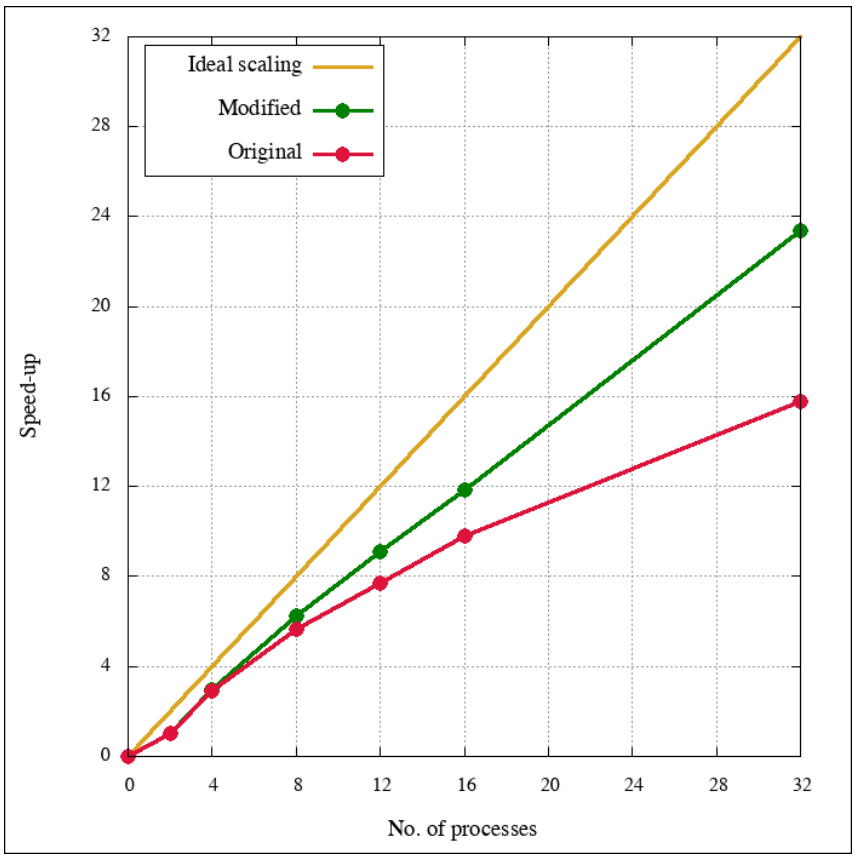

Figure 1: *The plot shows parallel speed-up against the number of MPI processes for both the original program and the modified version.*

version increases linearly with the number of MPI processes, while the speed-up for the original version starts to decelerate after 8 processes. The maximum speed-up achieved by the modified version was approximately 23 for 32 MPI processes on 2 nodes, which is approximately 50% higher, compared to the maximum speed-up of approximately 16 for the original version. By using a message buffer to send results from worker processes to the manager process after each column calculation, the program reduces the amount of data that needs to be communicated at once and allows the manager process to store the results in the nIter array as soon as they are available.

Conclusion

In conclusion, this report presents the modifications made to the communication pattern in the manager-worker version of the Mandelbrot program to optimize its parallel performance. The original code, which used MPI_Reduce to communicate results at the end of the calculation, was modified to have worker processes send their results to the manager process after each column was calculated. The results of the scaling study carried out on the Isca HPC system demonstrate that the modified program performs significantly better than the original version with 12 processes or more. This modification reduces the amount of data that needs to be communicated at once and allows the manager process to store the results in the nIter array as soon as they are available. Overall, these modifications improve the program's parallel performance and demonstrate the importance of optimizing communication patterns in parallel computing.